Code 1: Dynamic Time Wrapping

Trajectory	Distance
($X1$,$X2$)	26.31
($X2$,$X1$)	26.31
($Y1, Y2$)	68.11
($Y2$,$Y1$)	68.11
($Z1$,$Z2$)	9.58
($Z2$,$Z1$)	9.58

With the pointwise distance matrix, just looping start from i=0 and j=0; then the border; and at the end, apply the algorithm all other points.

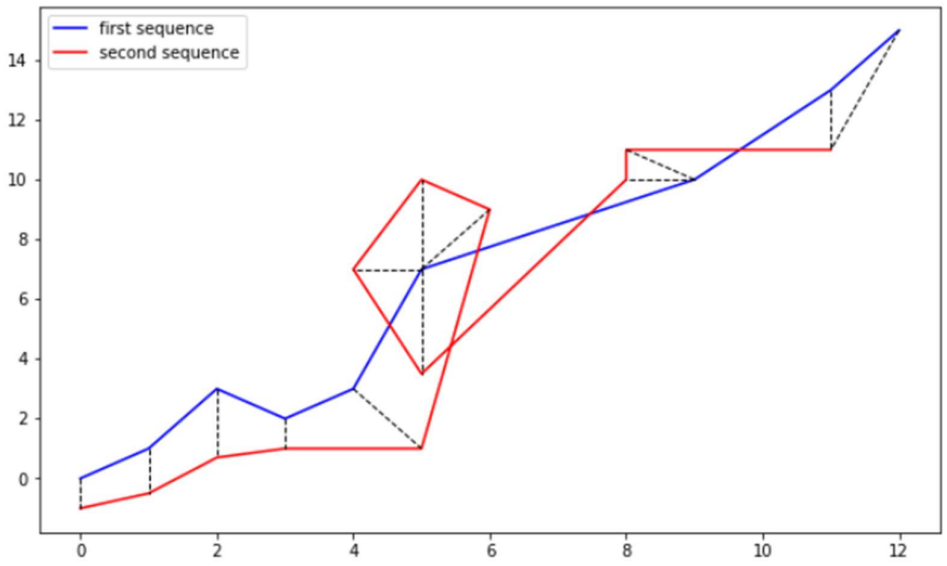
```
self.gamma[0][0] = dist[0][0]
#for two borders
for a in range(1, self.ny):
    self.gamma[0,a] = dist[0,a] + self.gamma[0, a-1]
for b in range(1, self.nx):
    self.gamma[b,0] = dist[b,0] + self.gamma[b-1, 0]

for i in range(1, self.nx):
    for j in range(1, self.ny):
        self.gamma[i][j] = dist[i][j] + min(self.gamma[i-1][j-1], self.gamma[i][j-1], self.gamma[i-1][j])
return self.gamma
```

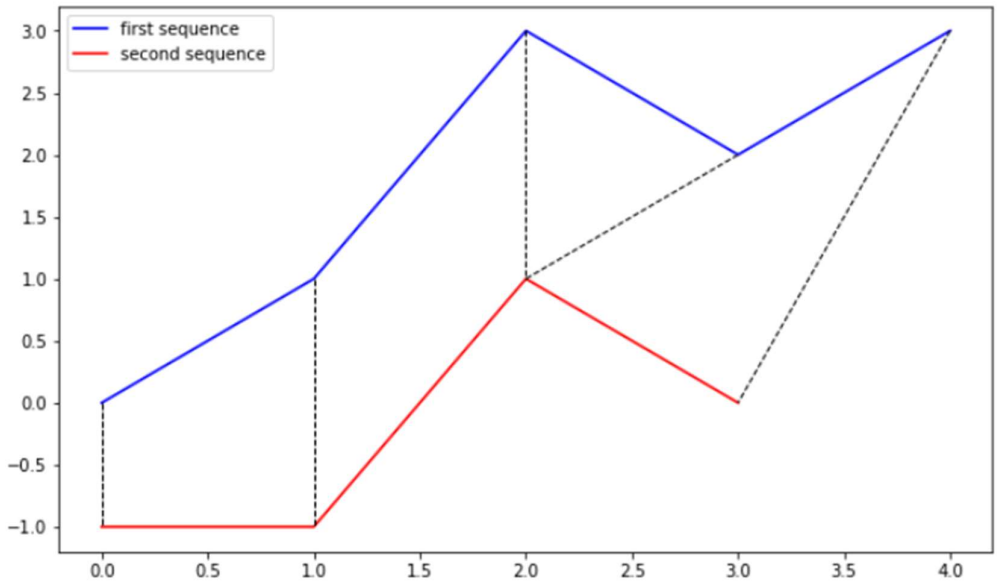
Backtrack (Bonus):

The basic idea is to trace back and to see the which value of the distance matrix corresponding to the gamma matrix.

- For X1 and X2:



- For Z1 and Z2:


first sequence
second sequence
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Code 2: KMeans ++

- **The basic idea of KMeans++**

Cluster centers are initially chosen at random from the set of input observation vectors, where the probability of choosing vector x is high if x is not near any previously chosen centers.

The cumsum here provides a way to choose center under certain probability boundaries providing something resembling the cumulative probability. This specific way of thinking credits to StackOverflow.

https://stackoverflow.com/questions/5466323/how-exactly-does-k-means-work

```
#=========================kmeans ++ ====================================
r = np.random.randint(0, len(X) - 1)
centers[0] = X[r]

for i in range(1,k):
    #calculate distances
    D2 = np.array([min([np.linalg.norm(c-
x) ** 2 for c in centers]) for x in X])

    probs = D2/D2.sum()
    cum = probs.cumsum()

    r = np.random.random()

    for j, p in enumerate(cum):
        if r < p:
            index = j
            break

    centers[i] = X[index]
```

- **Assign and Update**

The basic idea is to create a dictionary storing the corresponding data points to each clusters, like *{0: [data points]; 1: [data points]...}*.

Considering that each center is a numpy array, I used the following function to update the centers by calculating the mean features of all the data points in that particular center. There is one difference here: I add an new parameter, that is, old_centers, to the orginal function. The reason is pretty straight-forward: for iterating and updating purpose.

```
for point_set in clusters.values():
    if(len(point_set)) > 0 :
        sum_point = point_set[0].copy()
        for a in point_set[1:]:
            sum_point = np.add(a,sum_point)
        mean_point = np.asarray(sum_point) * 1.0/len(point_set)
        new_centers.append(mean_point)
    else:
        new_centers.append(np.asarray(old_centers[counter]))
```

- **Iteration**

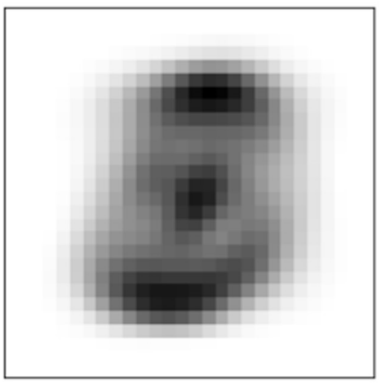
1. Assign the result from *assign_and_update* to new_centers.
2. Calculate the difference (maximum distances between old centers and new centers).
3. If the difference is larger than the tolerance, assign and update again until convergence.

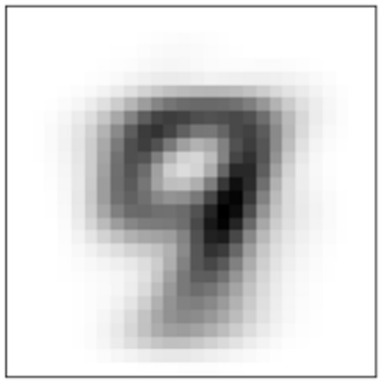
- **Result Showcase**

```
==================now K is : 2
1st:  1768.9181918040379
iteration: 1 1768.9181918040379
iteration: 2 150.1957908482931
iteration: 3 61.625546445548174
iteration: 4 40.37139333477302
iteration: 5 31.897018915495703
iteration: 6 26.73652351051441
iteration: 7 22.707507724879484
iteration: 8 19.909747937283996
iteration: 9 16.976964734491975
iteration: 10 13.880000559396015
iteration: 11 10.69945982953863
iteration: 12 8.786830689231632
iteration: 13 6.383036887017961
iteration: 14 5.4981082067541935
iteration: 15 3.8020980334445595
iteration: 16 2.563190219586549
iteration: 17 2.3028846487704024
```

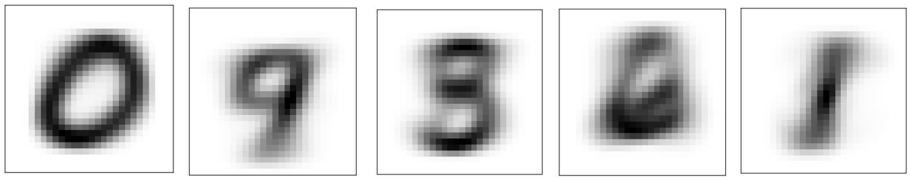
```
==================now K is : 5
1st:  1742.2406209170128
iteration: 1 1742.2406209170128
iteration: 2 245.5852658544085
iteration: 3 104.60010996047939
iteration: 4 59.50187976929675
iteration: 5 44.067775873653254
iteration: 6 43.67140647684688
iteration: 7 52.35780558124591
iteration: 8 70.68376734746319
iteration: 9 87.57631763512603
iteration: 10 92.18178885312108
iteration: 11 99.60945849417085
iteration: 12 103.07785773360715
iteration: 13 109.91288465880055
iteration: 14 126.11729495902254
iteration: 15 142.70923147267146
iteration: 16 148.82183712928
iteration: 17 133.47939027629042
```

4. K=2

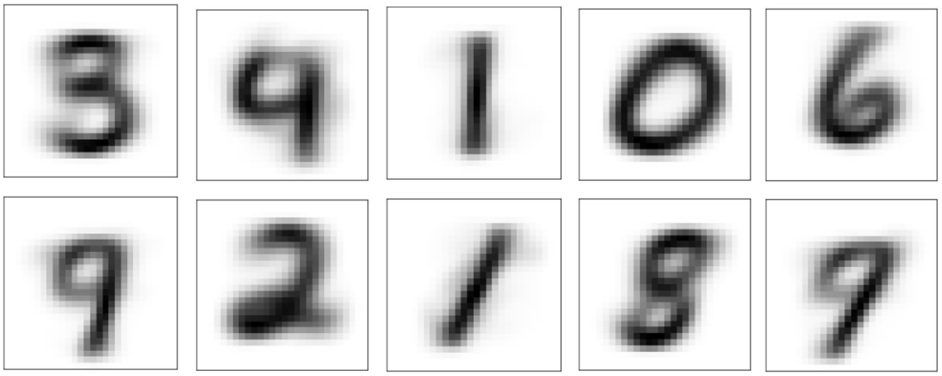




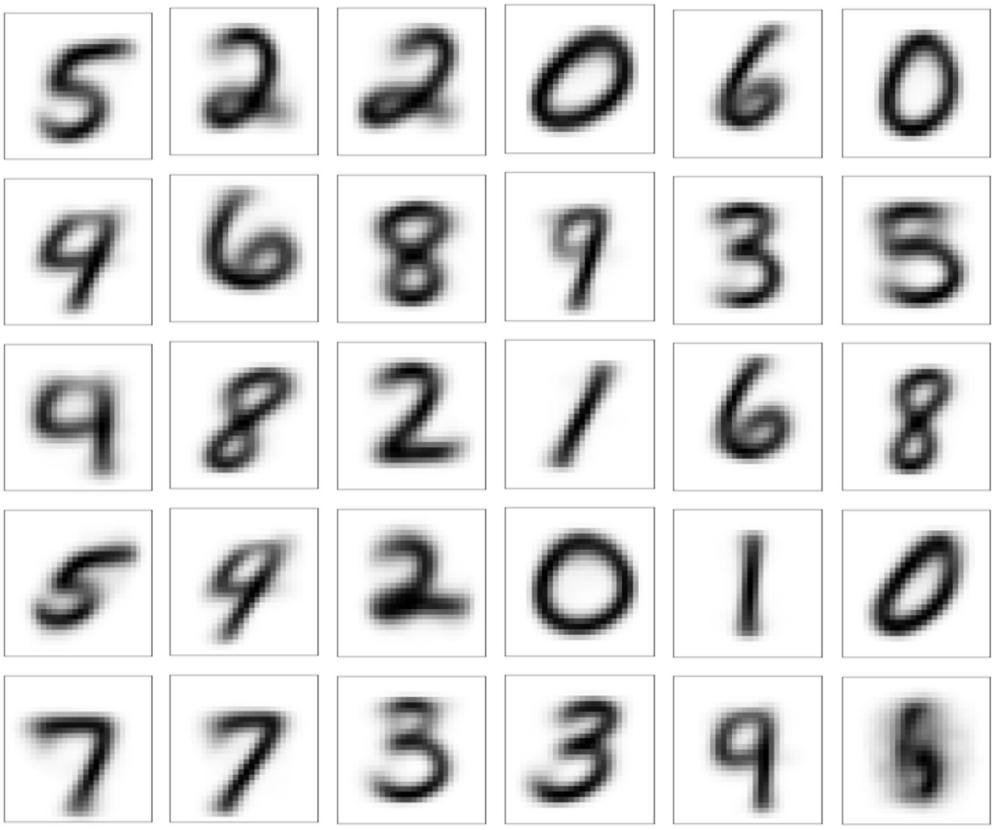
5. K=5



6. K=10



7. K=30



We could see that the clusters getting clear for k =2, 5, 10 and 30. For 2, it's gathering around 9 and 0. For k = 10, the clusters are basically from 0, 1, 2, …10. For k = 30, there are many same numbers.

Code 3: Clustering

- **KMeans:**

8. Fixed k = 100

for N = 100, the time is 0.581193s
for N = 1000, the time is 0.860646s
for N = 2000, the time is 1.315524s
for N = 4000, the time is 2.213522s
for N = 6000, the time is 3.114023s
for N = 8000, the time is 3.938611s
for N = 10000, the time is 4.557726s
for N = 20000, the time is 9.325501s
for N = 40000, the time is 23.910697s
for N = 60000, the time is 36.371281s
for N = 80000, the time is 51.893390s
for N = 100000, the time is 131.994945s

9. Fixed N = 10000

for k = 10, the time is 2.309763s
for k = 50, the time is 16.416492s
for k = 100, the time is 41.684083s
for k = 200, the time is 93.095096s
for k = 500, the time is 226.878303s
for k = 1000, the time is 449.588510s

- **MiniBatch, k =100**

10. Fixed k =100

for N = 100, the time is 0.122357s
for N = 1000, the time is 0.094840s
for N = 2000, the time is 0.098736s
for N = 4000, the time is 0.119681s
for N = 6000, the time is 0.251328s
for N = 8000, the time is 0.123670s
for N = 10000, the time is 0.352809s
for N = 20000, the time is 0.159576s
for N = 40000, the time is 0.224399s
for N = 60000, the time is 0.292260s
for N = 80000, the time is 0.348027s
for N = 100000, the time is 0.447805s

11. Fixed N =10000

for k = 10, the time is 2.828393s
for k = 50, the time is 16.882158s
for k = 100, the time is 42.547453s

for k = 200, the time is 88.619178s
for k = 500, the time is 229.841930s
for k = 1000, the time is 313.360983s

- **DBSCAN**

Try different eps values.

```
miles = np.arange(0.01,2.01,0.05)
kilometers = miles / 0.621371
EPS = kilometers / 100
```

eps	clusters	run time
0.000161	32	0.652733s
0.000966	59	0.734245s
0.00177	86	0.815868s
0.002575	96	0.911613s
0.00338	116	1.014884s
0.004184	135	1.131398s
0.004989	144	1.233594s
0.005794	151	1.421827s
0.006598	160	1.463858s
0.007403	148	1.568487s
0.008208	134	1.706298s
0.009012	114	1.814578s
0.009817	107	1.999963s
0.010622	96	2.197895s
0.011426	82	2.226279s
0.012231	75	2.444034s
0.013036	68	2.599751s
0.01384	62	2.909194s
0.014645	63	3.018279s
0.01545	63	2.986545s
0.016254	63	3.142756s
0.017059	61	3.216376s
0.017864	60	3.441132s
0.018668	63	4.048329s
0.019473	62	4.878664s
0.020278	64	4.248652s
0.021082	64	4.564935s
0.021887	63	4.547292s
0.022692	64	4.611600s
0.023496	63	4.777748s
0.024301	62	4.558660s
0.025106	63	4.854294s
0.02591	64	4.937961s
0.026715	66	5.094235s
0.02752	68	5.172543s
0.028324	69	5.219552s
0.029129	70	5.383627s
0.029934	70	5.704694s
0.030738	68	5.627625s
0.031543	69	6.069872s

When the eps value increases, the number of clusters detected by the DBSCAN increases as well. While when it reaches around 0.10, the number of clusters drops to around 70 and then remain unchanged.

- **Clustering System Design**

The data structure used here is Pandas Dataframe. The step could be divided into:

1. Run MiniBatch K-means and store the result of the MiniBatch K-means to a dataframe column as ‘cluster_Mini’.
2. For each datapoints for each MiniBatch K-means, run DBSCAN and store the assigned clusters to a dataframe column as ‘cluster_DB’
3. Check for the unique combinations of cluster_Mini and cluster_DB. Return the values of number of clusters.

- MiniBatch Cluster

```
exp = MiniBatchKMeans(n_clusters=100, init='k-means++',batch_size=1000).fit(X)
d = {'lng': X[:,0], 'lat': X[:,1], 'cluster_Mini': exp.labels_}

df = pd.DataFrame(data=d)
```

- DBSCAN

```
for i in range(100):
    subset = df.loc[df['cluster_Mini'] == i]
    length = len(subset.lng.values)
    data = np.concatenate((subset.lng.values.reshape(length,1), subset.lat.values.reshape(length,1)), axis = 1)
    res = DBSCAN(eps=eps, min_samples=100).fit(data)
    subset['cluster_DB'] = res.labels_
     df.loc[df['cluster_Mini'] == i, 'cluster_DB'] = subset['cluster_DB']
```

Generate 1610 unique clusters(exclude -1) using 17.913252 seconds.

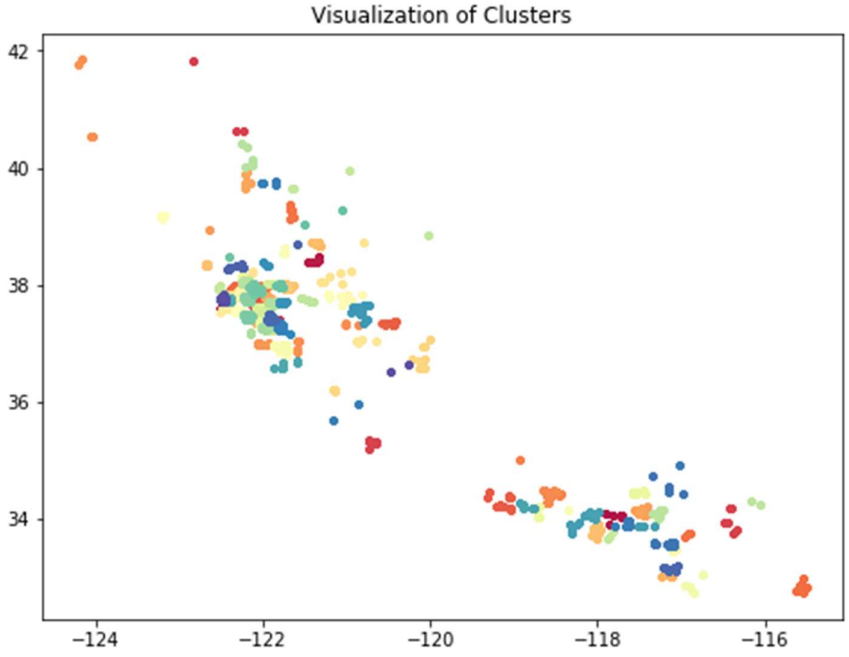
- Display:

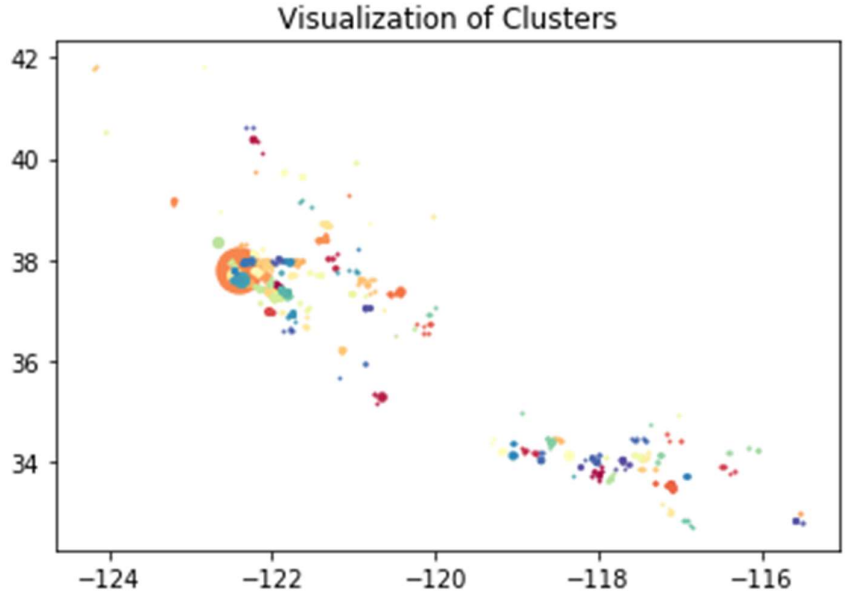
```
a = np.unique(df1['labels'].dropna())
unique_labels = np.append(np.unique(df1['labels'].dropna()),np.nan)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
#… other code
# and then…
for k, col in zip(unique_labels, colors):
    i +=1
    if k == np.nan:
        # Black used for noise.
        col = [0, 0, 0, 1]

    #class_member_mask = (labels == k)
    df2 = df1.loc[df1['labels'] == k]
    #xy = X[class_member_mask & core_samples_mask]
plt.plot(df2['lng'], df2['lat'], 'o', markerfacecolor=tuple(col), markersize=3)
```

Create distinct color for each cluster. Assign black color to those who is -1.

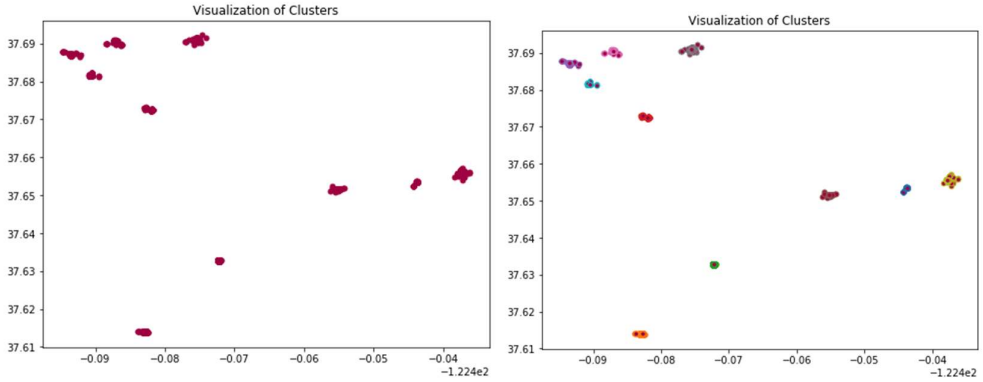
https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html#sphx-glr-auto-examples-cluster-plot-dbscan-py



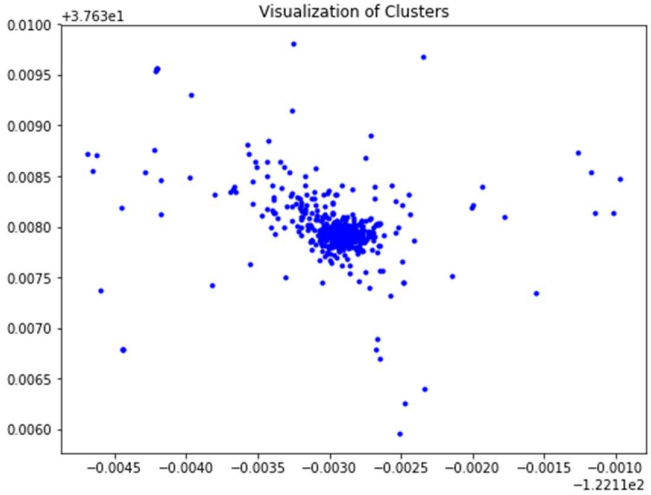


The latter plot is the plot according to the number of points in the cluster. We could find that there is one cluster having the largest number of points at the location at San Francisco.

The colors have trivial differences. If we plot only 10 clusters, the colors show little differences. If we change it to distinct color, we would see that data clustered for the 10 specific areas.



Let us investigate one typical cluster. The clusters have similar locations.



If we take a look at the text, we could notice that several twitters have similar context. They are from the same person at almost the same location.

969031	-122.112818	37.637917	13	25.0	13.25	@TooFadedShneel ok but come see me Bruh, wsup,...
969040	-122.113015	37.637740	13	25.0	13.25	@TooFadedShneel im dippin to the marines in Ja...
969044	-122.112971	37.637814	13	25.0	13.25	@TooFadedShneel s down to it ill fuck you up n...
968987	-122.112920	37.637896	13	25.0	13.25	This nigga can talk all he want, In the end be...
969000	-122.113074	37.637914	13	25.0	13.25	This nigga act like he started from the bottom...
71895	-122.112842	37.637870	13	25.0	13.25	Hahaha fuck no the mini a soulja
71930	-122.112842	37.637870	13	25.0	13.25	Fresh cut thanks to the one & only @jul
73836	-122.113066	37.637675	13	25.0	13.25	Baby lets do 69, hit me on my hotline
73880	-122.114204	37.639562	13	25.0	13.25	Tonight better be crackin cuh
73897	-122.114204	37.639562	13	25.0	13.25	Rockin with me
73904	-122.114204	37.639562	13	25.0	13.25	You rockin with the best
969074	-122.113024	37.638017	13	25.0	13.25	@Cubpimentel yap
969077	-122.112850	37.637914	13	25.0	13.25	Nope, I don't care. This nigga can catch this ...
969150	-122.113381	37.638138	13	25.0	13.25	@Cubpimentel ay nah we don't

If we choose a cluster with less data points…Yes, they are from the same person.

	lng	lat	cluster_Mini	cluster_DB	labels	text	user_id
99	-121.826334	37.324488	71	1.0	71.01	@kryzllicious I'm sooo gonna fail on that test...	594060435
13746	-121.826406	37.324360	71	1.0	71.01	I just want to stay in bed. 😩😩	594060435
31144	-121.826410	37.324417	71	1.0	71.01	I think I should get a haircut. 💁	594060435
31740	-121.826372	37.324535	71	1.0	71.01	What the fuuuuuck. 😂😭 @k00ladam #notcute http:...	594060435
31831	-121.826320	37.324431	71	1.0	71.01	@k00ladam #sonotcayute	594060435
33483	-121.826280	37.324466	71	1.0	71.01	@To_infinity04 You need to start reading those...	594060435
33915	-121.826358	37.324555	71	1.0	71.01	Believe me sweetie, I got enough to feed the n...	594060435
39674	-121.826246	37.324338	71	1.0	71.01	Ignorant ass people on ask.fm right now. 😖😒 #I...	594060435
42267	-121.826077	37.324687	71	1.0	71.01	Ugh. I give up on homework. 😒😒 #fuckdat	594060435
42413	-121.826329	37.324456	71	1.0	71.01	Who's going to the game tomorrow?	594060435
42448	-121.826299	37.324594	71	1.0	71.01	I want to go but like idk who to go with. 😐	594060435
66067	-121.826328	37.324538	71	1.0	71.01	idk if I should go to the game or not. 😕	594060435
67331	-121.826329	37.324457	71	1.0	71.01	I guess I'm not going to the game tonight. 😔 #...	594060435
76347	-121.826309	37.324404	71	1.0	71.01	It pisses me off how you ALWAYS need to be rig...	594060435
76694	-121.826267	37.324453	71	1.0	71.01	Why does it feel like it's Sunday? 😳	594060435